DESIGNING A PORTABLE ARCHITECTURE FOR INTELLIGENT PARTICLE ACCELERATOR CONTROL

W. Klein, C. Stern, Vista Control Systems, Inc., 134B Eastgate Dr., Los Alamos, NM 87544 and
G. Luger, E. Olsson, University of New Mexico, Albuquerque, NM 87131

Abstract

We present a portable system for intelligent control of particle accelerators. This system is based on a hierarchical distributed architecture. At the lowest level, a physical access layer provides an object-oriented abstraction of the target system. A series of intermediate layers implement general algorithms for control, optimization, data interpretation, and diagnosis. Decision making and planning are organized by knowledge-based components that utilize knowledge acquired from human experts to appropriately direct and configure lower level services. The general nature of the representations and algorithms at lower levels gives this architecture a high degree of potential portability. The knowledge-based decision-making and planning at higher levels gives this system an adaptive capability as well as making it readily configurable to new environments. Significant successes of this work are reported in [1, 2].

1 DESIGNING AN ARCHITECTURE

To be most useful, control architectures should be flexible enough to incorporate the important tools necessary for robust control, and should include design features which support the application of those tools in a timely and transparent manner. Our control architecture attempts to meet the following six requirements:

1. The use of conventional control techniques where appropriate. Conventional control is the best solution for a large class of problems, and consists of established, well developed, robust techniques for dealing with linear and approximately linear small systems.

2. The use of high-level control knowledge from experts in the field. More complex control systems include high-level knowledge and information obtained through knowledge-engineering sessions with a human expert. This knowledge is preserved in the form of symbolic data sets (rules, relations, objects, etc.) and is manipulated through sophisticated, high-level reasoning mechanisms. This knowledge-based component encodes internal control information about how and when to use classes of control algorithms and heuristics and for storing configuration information for PID, neural network, fuzzy control, etc.

3. The use of supervisory control for dealing with macro state transitions. For instance, in beam line tuning, if a failure in an upstream monitor forces the use of stripline data in judging beam intensity and distribution, a downstream controller may need to use a control algorithm which is less precise, faster, and less sensitive to intermittent failure or noise. Supervisory control can also be used for control over different internal control subsystems.

4. Support for real time reactive control. In this case, "real time" means the ability to compute and perform "satisfactory" or "good enough" decisions that are not delayed due to control system response.

5. The control of complex systems through the use of a hierarchical distributed architecture. To perform intelligent control that optimizes behavior in complex systems, a control architecture should support the ability to coordinate the individual partitions of the system to achieve an overall goal. This is especially true when controllers operate on platforms distributed throughout a plant. A distributed architecture requires structures for relaying information between nodes, synchronizing information, coordinating timing, etc. A distributed control system may also require global access to certain data structures including process models, diagnostic systems, alarm systems, and hardware.

6. Portability across classes of similar control problems. For a system to be truly portable, domain specific algorithms should work at many facilities with limited modification. The control system should have a working model of the domain that allows it to accomplish similar goals at numerous facilities with different specific hardware designs, but similar classes of control elements.

2 PHYSICAL ACCESS LAYER

The *Physical Access Layer*, or PAL, is what allows the control system access to hardware, portability, dynamic construction of representation, filtering of data, coordination of hardware, and conflict detection. The PAL is permanent and static, but hierarchical structures in the PAL can be built dynamically at run time. Learning can occur in the PAL in various feature detectors and pattern recognition components, as well as in recording attributes of specific physical devices.

The Physical Access Layer provides a mechanism for controllers and optimizers to access the physical system. Access to hardware is supported through a single layer for a number of reasons. First, the control system can be ported to multiple systems whose hardware varies in implementation detail but not functionality. A globally available, uniform interface to hardware keeps controllers and solvers from having to explicitly represent low-level handlers and hardware handshaking mechanisms. The PAL is one of the only shared global repositories of information in the control system. This makes it an ideal mechanism for conflict resolution.

The PAL also plays an important role in transforming raw data into a representation for higher level reasoning. This translation may be simple filtering of data, as in noise reduction, pattern classification, and error detection or it can be more sophisticated. In particular, the PAL is responsible for providing initial feature detection, fuzzification, data analysis, and discretization of information. This is particularly important in the context of a knowledge-based or symbolic system where raw numeric data is often inappropriate for manipulation. For instance, the PAL can be used to transform image data into a set of relevant image features, e.g., background noise, average pixel intensity, or existence of a defect.

The PAL is also useful for detecting, preventing, and resolving resource conflicts in a distributed control system. If an obvious resolution exists, the PAL provides coordination instructions to the conflicting units. Otherwise, the PAL can simply report the conflict and allow resolution to occur within the control system itself.

A further purpose of the PAL is to support an abstract translation mechanism between the low-level control system interface accessible through Vsystem channels [3] and the control architecture's multi-representational interface. The PAL accesses Vsystem directly through a library of Vsystem API calls encapsulated in a channel class library built using object-oriented programming techniques in C++. Simple channels are represented as objects with enhanced features including data validation, recording and retrieving defaults, time delayed action, equipment limit checking, and value stepping.

The PAL is designed for fast parallel operation. Each object that can receive messages from outside the PAL is run as a separate thread. Messages are received in a single message handler and immediately distributed to a message buffer associated with the recipient object.

PAL objects are not confined to representing physical devices. They may also be used to implement abstract devices defined as the collection and interacting behavior of a group of other objects. Abstract objects can be used to build control knobs that transform a single signal into a set of signals to many devices. Tuning knobs are sometimes used in accelerator control when the relationship of a set of quadrupoles and their effect on the beam at a downstream location can be determined by a series of non-linear functions. Tuning then occurs by manipulating a single knob to affect a single beam parameter while maintaining other beam properties, e.g., magnifying the beam without increasing divergence.

3 DISTRIBUTED COMPONENT CONTROL

Our system is based upon a distributed hierarchical architecture designed to incorporate a wide variety of representations, both analytical and knowledge-based, into a single control framework. At the heart of the architecture is a group of knowledge-based *controllers* that apply control from a local viewpoint. These controllers are hierarchically organized in a structural/functional hybrid design [4].

Controllers are responsible for making decisions about how control actions will be performed, what those actions will be, when they occur, and how their performance will be measured. Controllers are also responsible for reasoning about system state, diagnosing errors in control solutions, decomposing goals into tasks and actions, and initiating any necessary human interaction. Knowledge is encoded in each controller as a set of relations (n-ary predicates) and a set of operators that manipulate those relations. The predicates describe relationships between controller variables, facts about process state, task decomposition relationships, and message parameters for controller communication.

In addition to controllers that encode information about domain specific elements, algorithms, and experience, we also use *solvers* for component-based application of specific algorithms. Solvers encode pieces of algorithms which can be put together (again in a hierarchical manner) for run-time construction of control algorithms. Consider, for example, a search procedure for a search algorithm such as simple hill-climbing for optimizing beam transmission. The procedure is broken into three parts, a component for generating data points that must be measured during search, a component for measuring the data points using appropriate elements in the domain, and a parent controller which coordinates actions of the two children in a way that produces hill-climbing. A different algorithm, such as Newton's method, can be constructed by merely substituting the parent controller for one that applies a different top-level procedure. Different child controllers may also be substituted when operating different sets of control elements or in cases where specific constraints (e.g., noise handling, speed, etc.) are important [5].

4 TELEO-REACTIVE BEHAVIOR

Controllers built for tuning beam lines were designed to carry out reactive plans to maintain robustness and to support dynamic re-planning based on changing goals or world states. Reactivity was obtained by using a teleo-reactive (TR) rule structure. The TR architecture was originally designed as a control system for autonomous robots. [6] The purpose of teleo-reactive programs, also called TR trees, is to encode a plan for achieving a goal and allow reactive execution of control actions based on observed world state. TR plans are coded as ordered lists of condition-action pairs. Each condition specifies a state of the environment that must be true in order to perform the action. The TR plan is executed by checking the preconditions of each TR rule in the tree in order. If a rule has a satisfied precondition, its associated action is taken. World state is then reevaluated and the tree is cycled again. Only the rule with highest ordering whose conditions are satisfied is activated at each iteration.

Each control action is intended to make a condition for a higher ordered rule true. Control algorithms are provably correct if the union of all rule preconditions identifies the universe of possible world states, and if all actions guarantee the eventual satisfaction of a higher precondition. TR trees allow reactive plan execution, since the action that corresponds with the observed world state is taken at each time step. If the world state changes to a state further from the goal state, a lower level rule automatically activates and control moves the state closer to the goal. If a benevolent world state occurs, causing the world state to jump closer to goal, the control system skips unnecessary control actions.

5 LEARNING

Learning is intended to occur in our control architecture at many levels. Rather than regarding learning as a centralized activity, the framework supports the use of local learning techniques in different structures. This view of adaptive behavior fits in with the overall distributed local-control scheme on which the entire architecture is based [5].

Low-level learning in the form of pattern recognition and feature detection can occur in the PAL through the use of adaptive filters attached to feedback objects. Neural networks, fuzzy classifiers, and other adaptive systems can identify system behavior and store learned information as network weights, fuzzy sets, etc. These filter parameters can be stored at the controller level and passed back to the PAL during future tuning.

Conventional adaptive control methods can be used at the solver level to enhance the locally adaptive nature of the system. Well understood techniques exist for applying such algorithms as adaptive PID, Kalman filters, etc. Controller-level learning involves deeper knowledge-level issues. Since these modules operate over longer time scales and often contain strategic or goal oriented information, high level learning algorithms such as case based learning and decision tree induction are appropriate [5].

One area of learning which we have begun to study in detail is model refinement. Model refinement is the process of identifying system behavior with some uncertainty, and then refining the model through observation as well as active experimentation to reduce uncertainty or adapt the model to changing conditions. Physicists often use complex model refinement techniques to work from an analytic model, as coded in TRANSPORT, COSY, etc., and iterate between beam tuning and model adaptation and search. Our goals are to adapt these methods within the framework of our automatic control system.

5 SUMMARY

We have described a distributed, hierarchical architecture for beamline control combining heuristic, knowledge-based, and conventional control methods. This hybrid architecture integrates a variety of reasoning, search, and pattern recognition methodologies. Preliminary tests, reported in [1, 2] indicate the potential for emulating and often exceeding the performance of skilled human operators. Continuing research includes extending the diagnostic, model refinement, planning, and learning components of the system.

ACKNOWLEDGMENTS

This work was supported by a DOE SBIR contract (#DE-FG05-95ER81897) to Vista Control Systems, Inc. We thank Mike Kroupa, Bob Knight, and Bob Westervelt of Vista. We thank Andy Jason of LANL and Xijie Wang of BNL-ATF for helping us understand accelerator physics.

REFERENCES

[1] Klein, W., Stern, C., Kroupa, M., Westervelt, R., Luger G., and Olsson, E. 1997. Tuning and Optimization at Brookhaven and Argonne: Results of Recent Experiments, *Proceedings of the Particle Accelerator Conference*, American Physical Society.

[2] Klein, W., Stern, C., Luger, G., and Olsson, E. 1997. An Intelligent Control Architecture for Accelerator Beamline Tuning, *Proceedings of Innovative Applications of Artificial Intelligence*, Cambridge: MIT Press.

[3] Clout, P. 1993. The status of Vsystem. *Proceedings of the Third International Conference on Accelerator and Large Experiment Physics Control Systems*, Germany.

[4] Acar, L. and Ozgunner, U. 1993. Design of Structure-Based Hierarchies for Distributed Intelligent Control. In Antsaklis et al., *An Introduction to Intelligent and Autonomous Control*, Boston: Kluwer.

[5] Klein W. 1997. *A Software Architecture for Intelligent Control*, Ph.D. Dissertation, Computer Science Department, University of New Mexico.

[6] Nilsson, N. 1994. Teleo-Reactive Programs for Agent Control. *Journal of Artificial Intelligence Research* 1:139-158.